ASEAN J. Sci.Technol. Develop. 8(2)(1991) 53−76

THE DESIGN AND IMPLEMENTATION OF A PC-BASED SOFTWARE DESIGN TOOL

MASHKURI YAACOB*
Department of Electrical Engineering,
OW SIEW HOCK
Computer Centre,
University of Malaya, Kuala Lumpur, Malaysia.

(Received 19 June 1990)

ABSTRACT

The current software industry is facing numerous problems. As such, continuing emphasis has been given in finding ways to solve these problems with the main focus on improving software development productivity. This can be achieved by the application of computer aided software engineering (CASE) tools during the development process. This paper describes the development of a PC-based low cost CASE prototype software design tool to assist software designers and programmers in designing their software systems and programs. The tool is in C and shell commands and is developed on the PC/AT microcomputer running under XENIX System V, a derivative of the UNIX Operating System. The useful features, utilities and file system structure of the XENIX Operating System are fully made use of in the development of the tool.

INTRODUCTION

The importance of software engineering has been acknowledged since the late 1960's[1]. Research has shown that the amount of cost involved in the development of computer software is increasing tremendously. Figures from the United States for the year 1989 illustrate that approximately US$ 150 billion was spent on computer systems and that figure is projected to be US$ 230 billion by 1992. Out of these figures, $ 23.7 billion was spent on computer software and this is expected to grow to $ 37.5 billion by 1992[2]. Further, there has been a noticeable tremendous growth in software needs during the past two decades[3]. It is predicted that the demand for software will grow at a higher proportion in the next decade. In the same light, problems do exist and more often than not, most of the software products are being completed late, perform unsatisfactorily, require major changes during the implementation stage and are incompatible with the rapidly changing computer technology[4]. Despite all these, the software industry must meet the growing demands for more software[5].

Software engineering methodologies which incorporate systematic approaches to software development are now advocated as one of the tangible means to overcome the many varied problems of developing good software. A combination of these techniques and tools, now better known as CASE is seen as the viable means to support and facilitate the software development process. Some of these tools are diagramming tools, code generators, documentation generators and reverse engineering tools[6]. Examples of CASE tools which are currently widely available on PC's and workstations include Excelerator[7], Information Engineering Workbench/Workstation (IEW/WS)[8], and Interactive Developments (IDE)[9], which are developed based on the more advanced computer technology and thus they are expensive (by Asian Standards) CASE tools. For instance, Excelerator costs US$ 9,500 while IEW/WS costs more than US$ 4,500.

This paper describes the development of a prototype interactive software design tool (ISDT) on a PC. The development of the tool is based on simple and appropriate software engineering techniques and tools. The paper aims to illustrate the design and implementation of a tool for designing structure charts (SC)[10,11] and data flow diagrams (DFD)[10,12,13]. The SC and DFD tools were then used to design further supportive tools. The experience in developing ISDT around the XENIX operating systems[14], i.e. harnessing the shell commands to good effect, is also highlighted. It can be claimed that a useful tool such as that offered by ISDT can be built with a modest hardware set-up offering moderately powerful features to the software developer. The paper also aims to highlight the experience in developing a large PC-based software for the benefit of researchers in the region. The design and implementation of the tool took advantage of existing facilities of XENIX and illustrates the incorporation of the modules in the tool. It is hoped that this paper will stimulate further work in the development of PC-based software to enhance further existing facilities already made available by the vendors at very high expenses.

FEATURES OF THE TOOL

Basically, ISDT consists of five main separate and independent modules. These five modules are collectively known as ISDT-Commands, Data-Flow, Structure-Chart, Help and Miscellanea. These modules are controlled by vsh (visual shell) of the XENIX Operating System[15] which invokes and executes them when a user selects certain functions performed by these modules. The memo of the vsh[14,15] is customised to suit the various functions performed by ISDT. This vsh is invoked by a main program at the XENIX command level which acts as an entrance point into the visual shell.

The five major modules of the organization structure of ISDT are repetitively decomposed into their smallest manageable sizes so that the coding process for these modules can be developed easily within a short period of time. The five main modules of ISDT are independent from each other, as shown in Figure 1. The breaking down of each of the five main modules into its smallest manageable modules are also as independent from each other as possible so that low coupling is achieved. Each of the smallest modules is designed to perform one single task which is a criterion of cohesion. The breaking down of these modules into their individual smallest modules are illustrated by the hierarchical block diagrams of Figures 2,3,6 and 7 respectively.

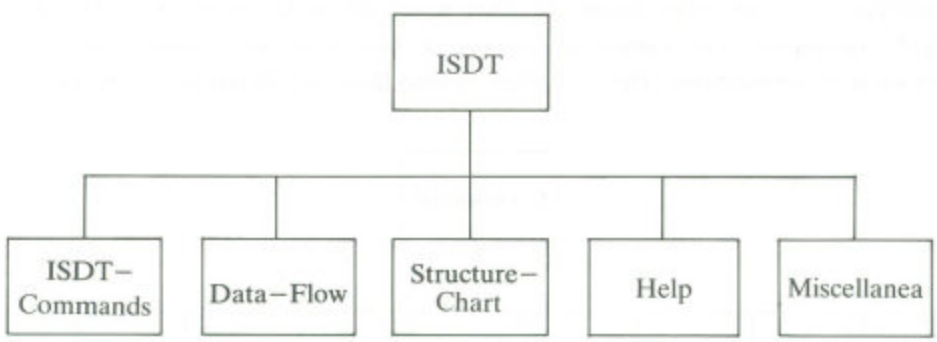


Fig. 1 The Organisational Structure of ISDT

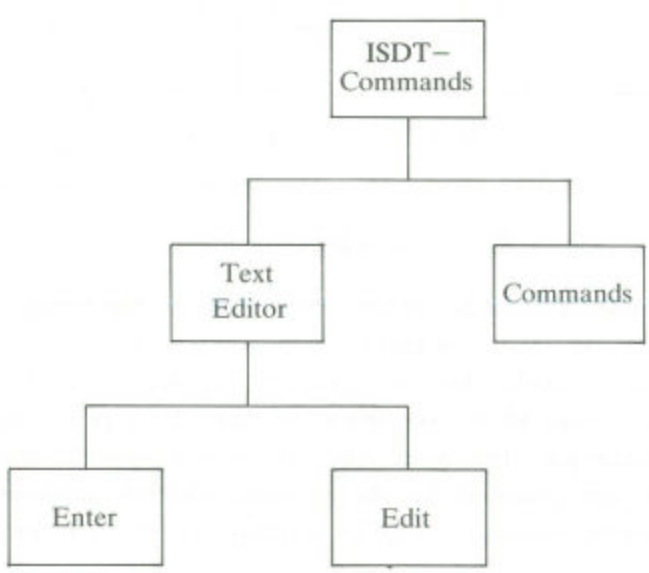


Fig. 2 The ISDT-Commands Module

Functional Description of Modules

The ISDT-Commands Module is for the ISDT user to develop or edit the DFD or structure chart of a *new* or an *existing* project. It provides the user with all the commands that are necessary for the development process of a DFD, structure chart, hierarchical block diagram or Hierarchy-Process-Input-Output (HIPO) diagram[5,16,17]. The command used to label the Jackson's Structured Programming (JSP) symbols is also included in this module[4,11,18].

This main module consists of two submodules, namely ***Text Editor*** and ***Commands***. The ***Text Editor*** submodule handles the entering or editing of labels. In other words, the ***Text Editor*** is used to handle the *Enter* and *Edit* command. On the other hand, *the* ***Commands*** submodule deals with the other ISDT commands. The ***Commands*** submodule can be broken down further into still smaller submodules. These smallest submodules are illustrated in Figure 3.

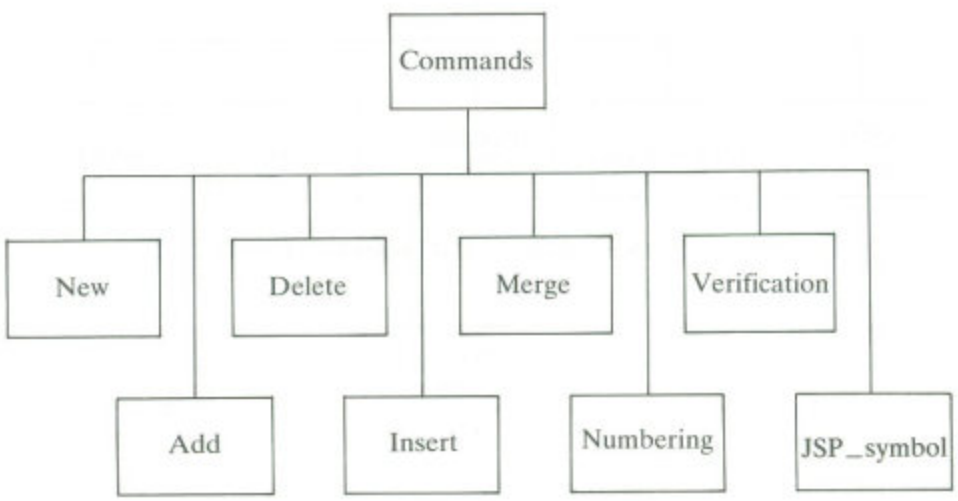


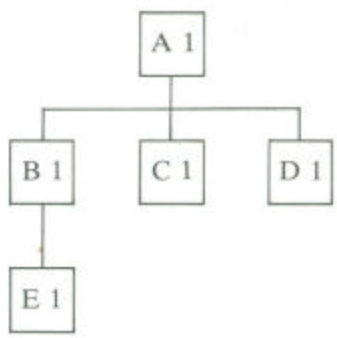
Fig. 3 The Commands Submodule

These commands include *New, Add, Delete, Insert, Merge, Numbering, Verification* and *JSP_symbol*. All these commands can be performed by the XENIX system commands in a combination of sequence, selection and iteration manner of execution. These ISDT commands are thus developed using the shell programming technique. The *New, Add, Delete and Insert* commands, as their names imply, are provided for the creation, addition, deletion and insertion of diagrammatic notations to DFD, structure charts or hierarchical block diagrams.

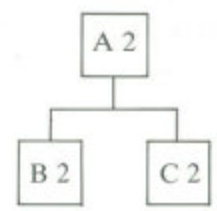
The Merge Module is to allow a user to merge two different structure charts or hierarchical block diagrams to become one single structure chart or

hierarchical block diagram. This module provides to the user two choices of merging. The first choice is to allow the user to merge a structure chart or hierarchical block diagram to the *left side* of a box or block of another structure chart or hierarchical block diagram. The second choice is to allow the user to merge a structure chart or hierarchical block diagram to the *end* of a box or block of another structure chart or hierarchical block diagram. The user is allowed to perform the two choices of merging at any level. These two choices of merging are illustrated in Figure 4.

Before Merging

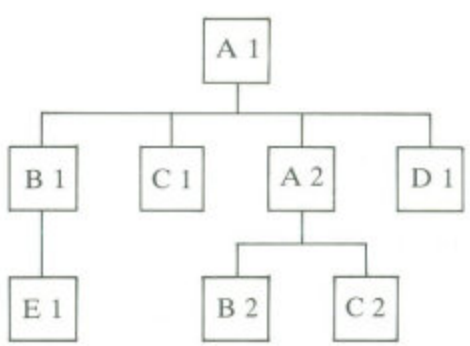


Structure Chart 1

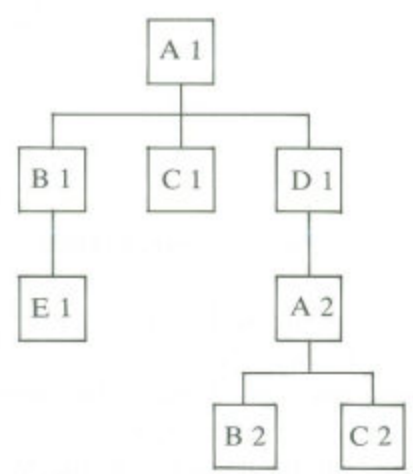


Structure Chart 2

After Merging



a. Merge to the left side of node D1



b. Merge to the end of node D1

Fig. 4 The Merging Process

The Numbering Module is designed to enable a user to number the structure chart or hierarchical block diagram. The numbering system is in accordance with the level numbers. For instance, at the first level, the only box or block is numbered as 1. At the second level, the boxes or blocks are numbered as 1.1, 1.2, 1.3 etc. Similarly, at the third level, the boxes or blocks extended from the boxes or blocks of 1.1, 1.2 or 1.3 are numbered as 1.1.1, 1.1.2, 1.1.3 etc, 1.2.1, 1.2.2, 1.2.3 etc or 1.3.1, 1.3.2, 1.3.3 etc. An example of this numbering system is illustrated in Figure 5.

The user is also allowed to unnumber the boxes or blocks when the numbers are no longer needed. The main purpose of this numbering system is to enable a user to trace and analyze the structure chart or hierarchical block diagram easily.

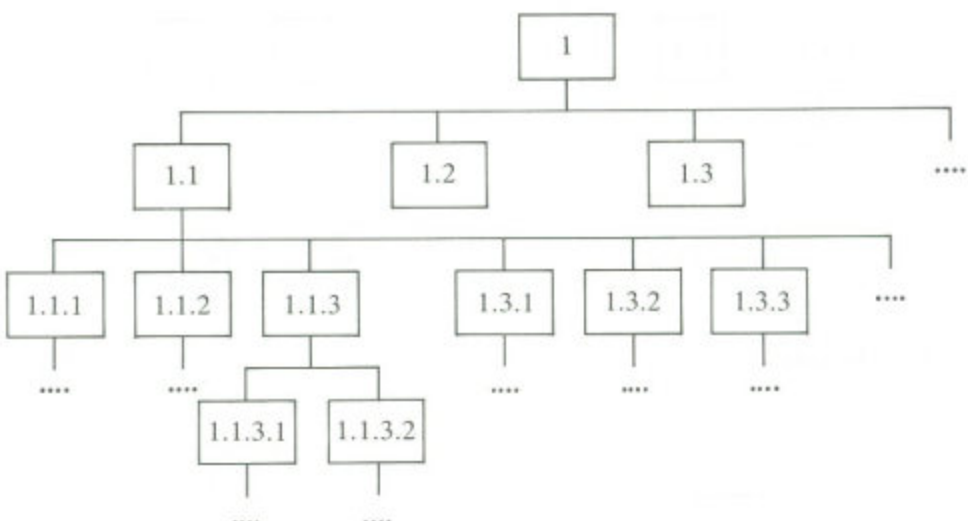


Fig. 5 The Numbering System

The Verification Module is to provide the checking facility on the duplicates of labels of a structure chart or hierarchical block diagram. The user will be informed of the verification result if duplicates of labels occurred in the structure chart or the hierarchical block diagram. A print out which shows the names and levels of the particular boxes or blocks having the same labels can be obtained instantaneously when requested by the user. The user can then edit the labels by referring to this hardcopy and selecting the *Edit* command.

The JSP_symbol Module is to enable the user to design his programs using Jackson's Structured Programming technique. In this technique, three types of symbols are used. They are the *sequence* (notation : none), *selection*

(notation : o), and *iteration* (notation : *). As there is no notation used to denote the *sequence*, it is not necessary to implement the symbol for *sequence* and thus only the *selection* and *iteration* notations are provided by this module. The user can label and edit these two/three notations easily. This is the module which is designed to support the program design process.

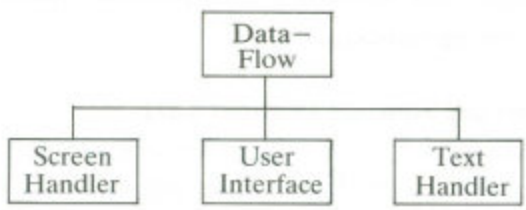


Fig. 6 The Data-Flow Module

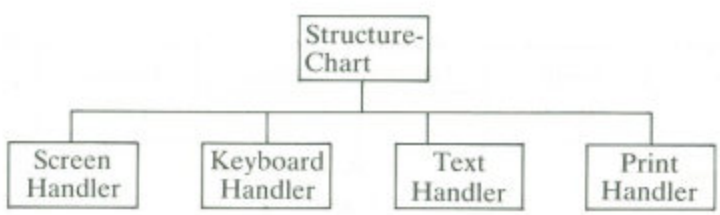


Fig. 7 The Structure-Chart Module

The Data-Flow Module is designed to handle the display of DFD on the terminal screen. It also enables a user to save the screen display to a file and to obtain a print out of the screen display instantaneously. This module consists of three submodules. They are the ***Screen Handler***, ***User Interface*** and the ***Text Handler***. The ***Screen Handler*** manipulates the display of DFD on the terminal screen ; the ***User Interface*** manipulates the functions to be performed after receiving a response from the user ; and the ***Text Handler*** manipulates the display of text within the DFD symbols. This module is developed using a combination of C programming and shell programming techniques.

The Structure-Chart Module is designed to enable a user to view the developed structure chart or hierarchical block diagram on the terminal screen. A user is allowed to view these diagrams a portion at a time by pressing the appropriate *up, down, left* and *right arrow keys*, and the *end* and *page down keys*. In addition, a user is also permitted to view these diagrams by choosing any one of the three sizes of diagrams; small, medium or large to be displayed. A print out can be obtained instantaneously if it is requested by the user.

The submodules of this main module thus consist of the ***Screen Handler*** which handles the screen display of the boxes or blocks in three sizes; ***Keyboard Handler*** which handles the functions to be performed when the user presses a specific key from the keyboard; **Text Handler** which handles the display of the text within the structure chart or hierarchical block diagram; and **Print Handler** which manipulates the printing of the structure chart of hierarchical block diagram. This module is also developed using a combination of C and shell programming techniques.

Data Flow Diagrams and Structure Charts of ISDT

In this section, sample data flow diagrams are drawn to illustrate the data flows of the modules of ISDT. They are the Data-Flow and Structure-Chart modules as shown in Figures 8 and 9 respectively. The corresponding structure charts are as shown in Figures 10 and 11 respectively.

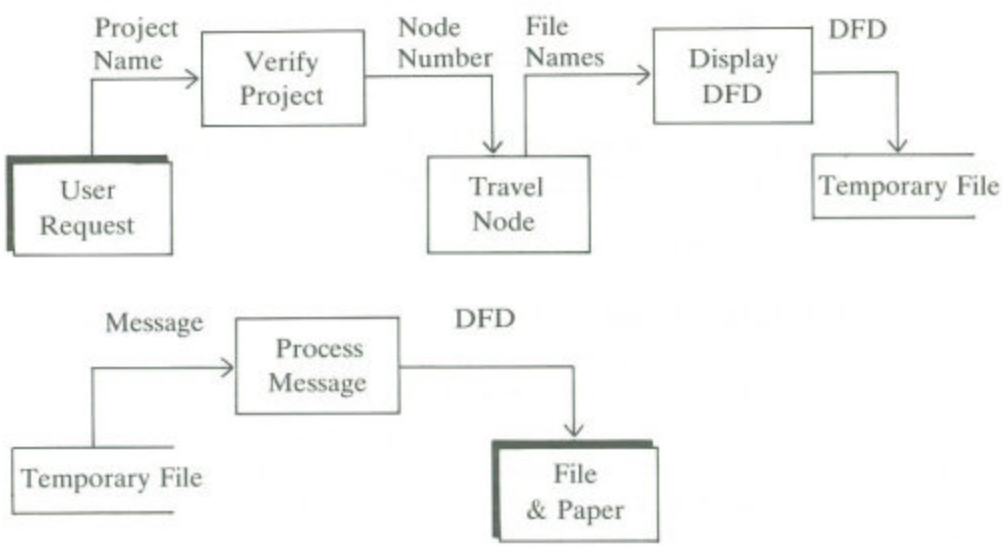


Fig. 8 The DFD of the Data-Flow Module

When a user makes a request to display a DFD, firstly a *Project Name* must be selected. This *Project Name* is then verified by the processing node *Verify Project*. Each of the *Node Number* under this project is passed to the processing node called *Travel Node*. These nodes are then travelled once so that information can be obtained from the *File Names* which are associated with each of the *Node Number*. This information is then passed to the processing node *Display DFD* which will display it in the form of *DFD* on the screen and at the same time save it in a *Temporary File*. A *Message* is

then output to the screen and is manipulated by the processing node called *Process Message*. The *DFD* displayed on the screen is then saved in a *File* defined by the user and/or printed on paper if a print out is requested by the user.

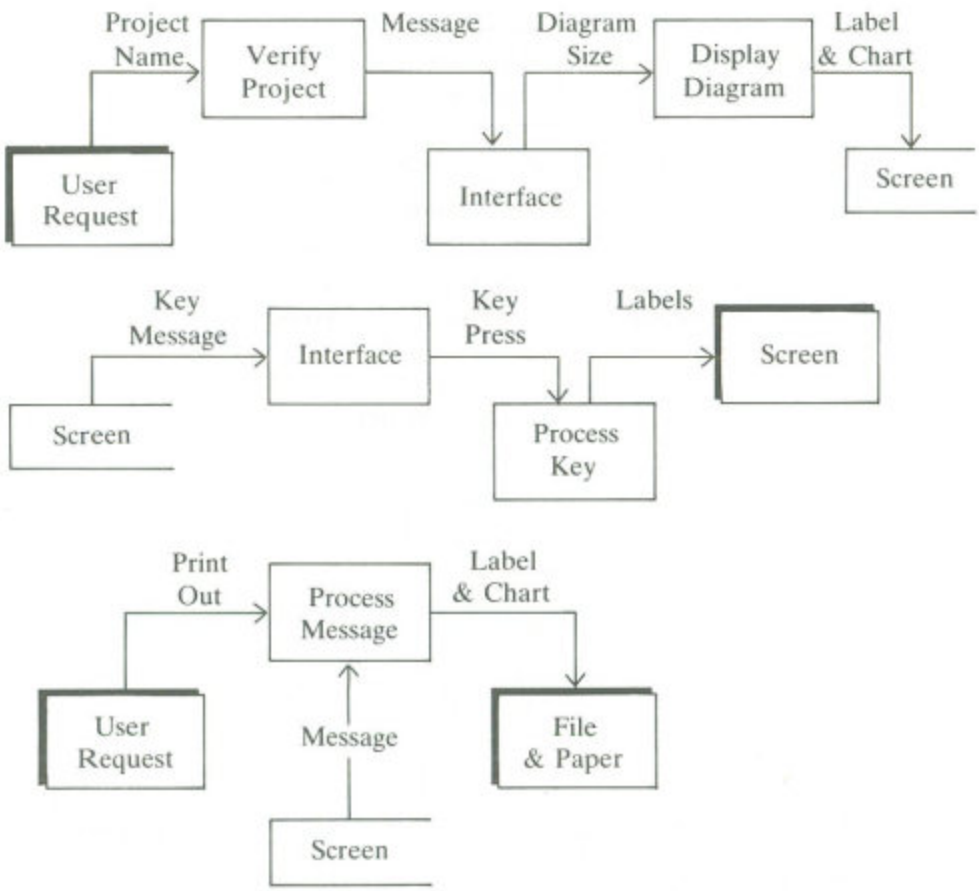


Fig. 9 The DFD of the Structure-Chart Module

When a request is made by a user to display a structure chart, a *Project Name* needs to be input and verified by the processing node *Verify Project*. A *Message* is then output to the screen and processed by the processing node called *Interface* so that the *Diagram Size* selected by the user can be passed to the processing node called *Display Diagram*. This processing node will display the *Label & Chart* on the *Screen* and at the same time *Key Message* is displayed and processed by the *Interface* processing node. When a key is pressed, it is manipulated by the processing node called *Process Key*. This processing node will update the display of the *Labels* of the

boxes on the *Screen*. If the user requests for a *Print out* of the diagram, a printing and saving *Message* will be displayed on the screen and then processed by *Process Message* to produce the Label & Chart which are then saved in a *File* and printed on *Paper*.

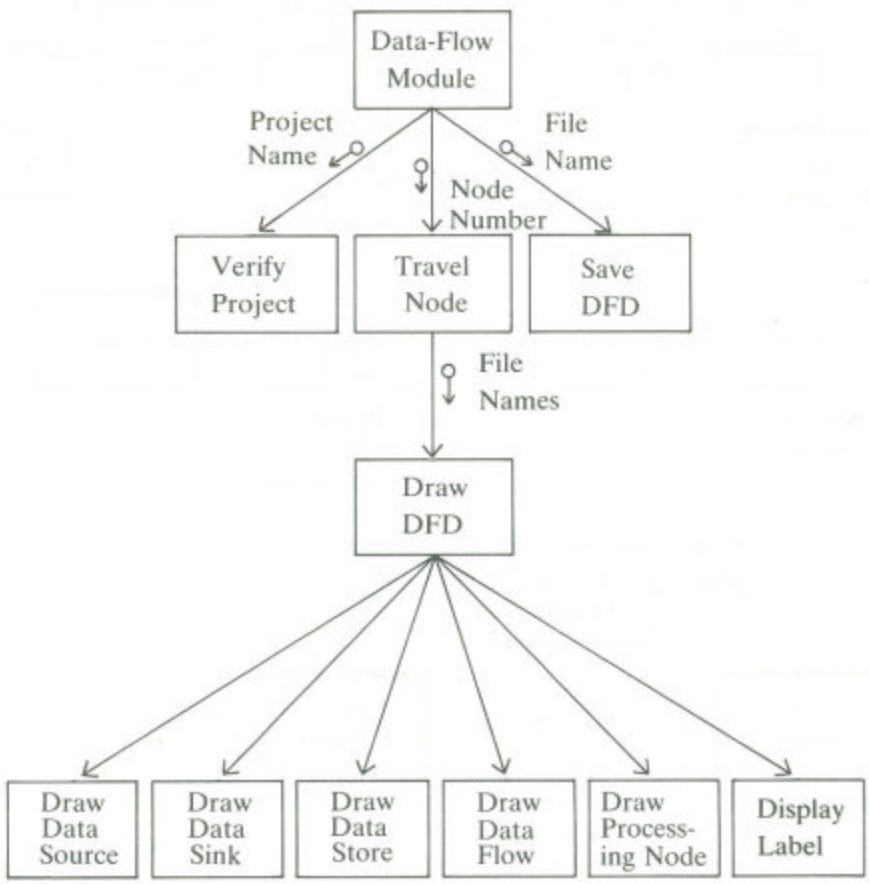


Fig. 10 The Structure Chart of the Data-Flow Module

The *Data-Flow Module* calls the *Verify Project* module and sends the information *Project Name* to this module. After the *Project Name* is being verified, each of the *Node Number* under this project is then sent to the *Travel Node* module to travel all these nodes. During the travelling node process, the information *File Names* associated with the nodes are sent to the *Draw DFD* module in which a DFD is drawn on the screen by calling the *Draw Data Source, Draw Data Sink, Draw Data Store, Draw Data Flow, Draw Processing Node* and *Display Label* module. A *File Name* defined by the user is sent to the *Save DFD* module when it is called to save the DFD drawn on the screen.

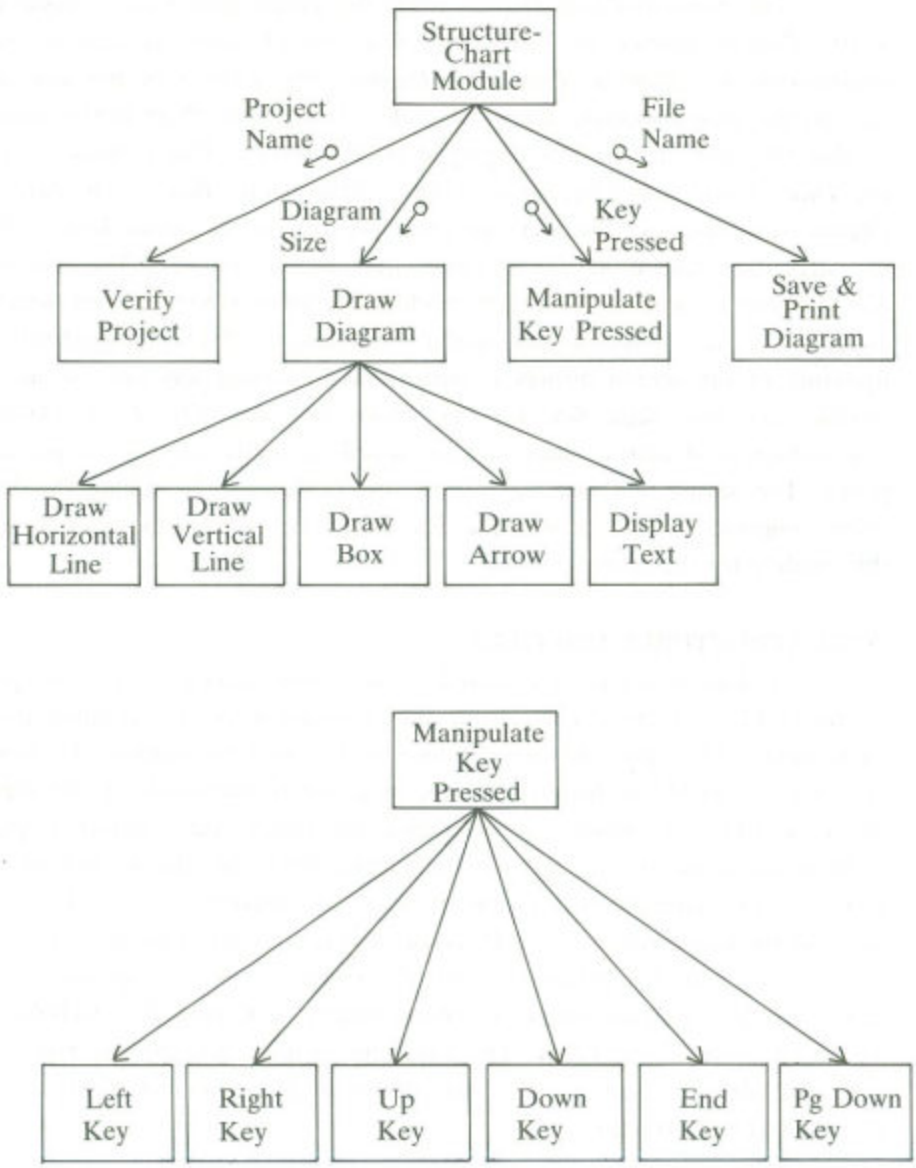


Fig. 11 The Structure Chart of the Structure-Chart Module

The *Structure-Chart Module* sends the information *Project Name* to the *Verify Project* module to verify that this *Project Name* is created for the construction of a structure chart. The *Diagram Size* selected by the user is then sent to the called module *Draw Diagram*. A structure chart drawn according to this *Diagram Size* is thus displayed on the screen. This is done by calling the *Draw Horizontal Line*, *Draw Vertical Line*, *Draw Box*, *Draw Arrow* and *Display Text* modules. The user can press any one of the arrow keys to browse the structure chart if its size is larger than the screen size. The information *Key Pressed* is sent to the called module *Manipulate Key Pressed* which will process the *Key Pressed* and update the screen display accordingly. The updating of the screen display is performed by calling any one of the called modules *Left Key*, *Right Key*, *Up Key*, *Down Key*, *End Key* or *Pg Down Key*. The complete structure chart can be saved in a file as well as printed on paper. The saving and printing process are performed by calling the *Save & Print Diagram* module in which a *File Name* defined by the user is sent to this module for the saving process.

DATA STRUCTURES AND FILES

A data structure is required to store information on the components of the DFD, structure chart, and the association of the components with each other. This data structure is also to be used to support the functions provided by ISDT. A hierarchical tree structure is employed for the representation of DFD, or structure chart, during the design and construction processes of these diagrams. It is a structure which can exhibit and offer an understandable and effective representation of the state of the application of ISDT. Further, the construction process of a DFD or structure chart requires a number of files to be created in order to store the labels for these diagrams. Some of these files are assigned unique extension names to denote the symbols associated with each of the labels. The following sections describe the type of files used and the representations of the different diagrams and charts using the hierarchical tree structure.

The Files Used

To construct a DFD, a number of files need to be created. The names and functions of these files are listed in Table 1.

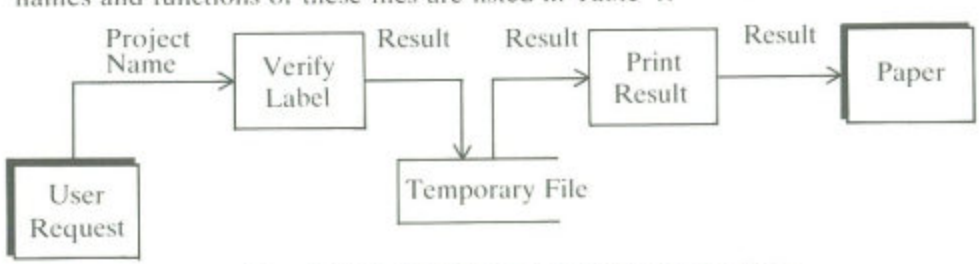


Fig. 12 The DFD of the Verification Module

TABLE 1
THE LIST OF FILES CREATED FOR THE CONSTRUCTION OF A DFD

File	Function
text.LA	to store the **la**bel (description) of a DFD.
text.SO	to store the label for data **so**urce symbol.
text.SI	to store the label for data **si**nk symbol.
text.ST	to store the label for data **st**ore symbol.
text.NO	to store the label for processing **no**de symbol.
text.FL	to store the label for data **fl**ow symbol.

However, the construction process of the structure chart, hierarchical block diagram or HIPO diagram only consists of the file named ***text*** which is created at each level for each box or block. It is used to store the label of the box or block. Besides these files, there are two other files which need to be created for the construction processes of DFD, structure chart, hierarchical block diagram of HIPO diagram. These two files are named ***.type*** and ***.next_son*** :

TABLE 2
THE FILES CREATED FOR THE CONSTRUCTION OF DFD
AND STRUCTURE CHART

File	Function
.type	to store the type of charts developed, that is, DFD or structure charts (SC); the contents of this file is either ***DFD*** or ***SC***.
.next_son	to keep track of the number of nodes and text files in each lower levels.

The Representations of the Diagrams and Charts

The construction of a DFD, structure chart, hierarchical block diagram or HIPO diagram is fully based on the hierarchical tree structure. The DFD of Figure 12, for instance, is developed using the strategy illustrated in Figure 13.

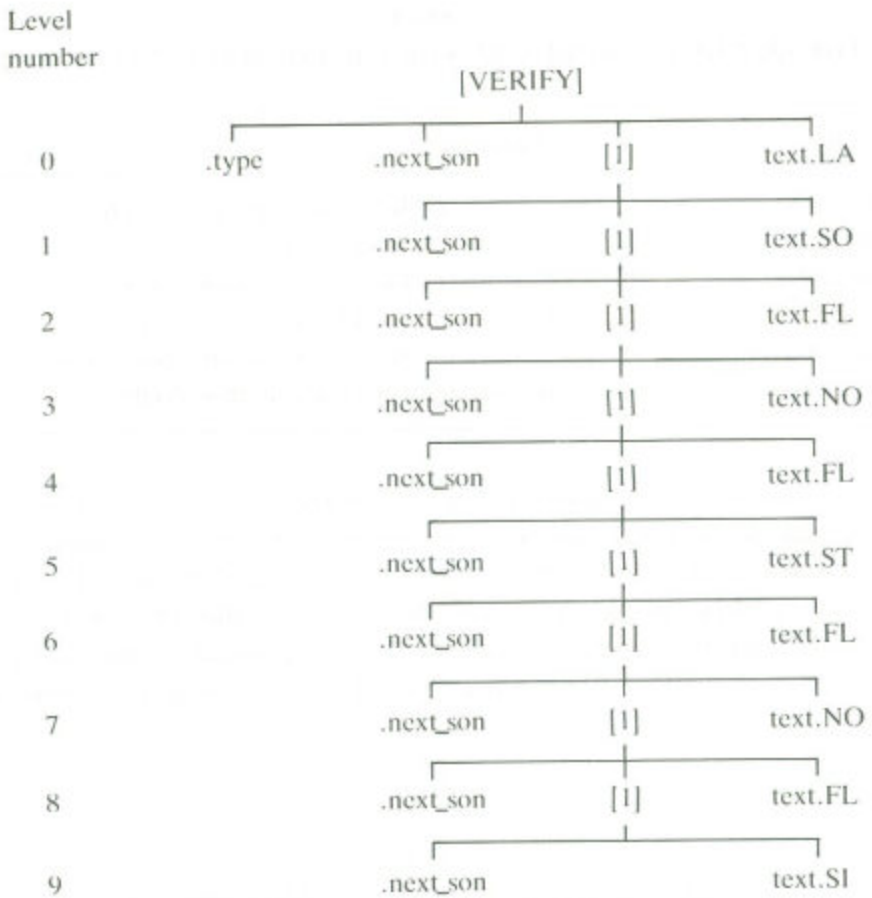


Fig. 13 The Representation of a DFD (Figure 12 Using Hierarchical Tree Structure)

The symbol [ ] is used to denote a *directory* or *node*. The text within the [ ] symbol is a *Project Name*. In this case, the *Project Name* is **VERIFY.** The sub-directories created below this level are considered as **nodes.** Each node is assigned a number. The node number is in ascending order when there are more than one node created at a level. The contents of the files created are summarised as follows (This summary should be compared with the DFD of Figure 12):

TABLE 3
THE CONTENTS OF THE FILES LISTED IN FIGURE 13

Level no.	Filename	Contents
0	.type	DFD
	.next_son	2
	text. LA	Fig. 12 : The DFD of the Verification Module
1	.next_son	2
	text.SO	User Request
2	.next_son	2
	text.FL	Project Name
3	.next_son	2
	text.NO	Verify Label
4	.next_son	2
	text.FL	Result
5	.next_son	2
	text.ST	Temporary File
6	.next_son	2
	text.FL	Result
7	.next_son	2
	text.NO	Print Result
8	.next_son	2
	text.FL	Result
9	.next_son	1
	text.SI	Paper

At each level (except level 9), there is a corresponding text file of extension *.LA*,.*SO*.,*FL*, or *.ST* and a node *[1]*. As such, the contents in the file *.next_son* at each level is 2. As there is only one text file of extension *.SI* at level 9, the contents of the file *.next_son* is thus 1. The contents in the file *.type* is **DFD** to denote that the nodes created under this project are used for the construction of a DFD.

IMPLEMENTATION DETAILS

The development process of ISDT is based on the top-down implementation of the **Main Module** downwards. The coding process can thus be done separately, and each module tested individually. After being tested for logic errors, they are then integrated and tested again. The purpose of this *unit testing* of the smaller modules in isolation is to avoid the occurrence of inevitable defects which may show up during the *integration testing* stage when a large collection of untested modules are linked together[5].

Main Module and ISDT-Commands Module

The major function performed by the **Main Module** is to create a *project* subdirectory and invoke the **vsh** which will display a selection menu to enable a designer or programmer to choose a particular function to be executed by one of the modules mentioned above. Since the algorithms and pseudocodes for the **Main Module** and some of the **ISDT-Commands Module** are simple and trivial, they are not discussed here. In this paper, only selected examples are included to illustrate some of the implementation features.

The Merge Module

The processes involved in the merging of two projects can be classified into the following three tasks :

Task 1 :

The first task is concerned with the tree traversal : inorder, preorder, postorder (Figure 14) of the project that will be merged to another project[19].

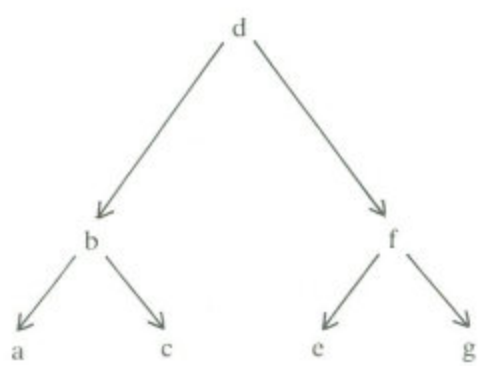


Fig. 14 Tree Traversal (Inorder, Preorder, Postorder)

The type of tree traversal chose the *postorder* method which is more suitable than the other two methods because the merging process also involves copying and deletion of files and sub-directories. The files and sub-directories of the tree structure can only be deleted when the travelling of the tree has returned to the parent level. During the postorder travelling process, the tree structure of the first project is also being created in the second tree structure of the project to be merged to.

Task 2 :

When the travelling of the tree has reached the leaves, all the files in that node are then copied to the corresponding node (sub-directory) that has just been created during task 1. On completion of the copying task, the process will return to the parent node.

Task 3 :

Task 3 is merely the deletion of all the files and/or sub-directories in the nodes that have all been copied over to the project to be merged to. This deletion task is executed by the shell command **rm.** The processes of task 1, task 2 and task 3 take place alternately until all the nodes have been travelled. The process of merging is then accomplished.

The Numbering Module

The processes involved in the **Numbering Module** can be classified into the following three tasks :

Task 1 :

Task 1 is similar to the first task of the **Merge Module.** The main purpose of this task is to obtain the pathnames of all the nodes.

Task 2 :

When the travelling of the tree has reached the leaves, the shell command **pwd** is used to get the current path. This pathname is then converted to a number which is then used to number the particular box at that level. The conversion process is as follows :

If the pathname is /ISDT/project/Proj_name/1/2/2/3, firstly the string /ISDT/project/ is removed from the pathname by a simple and general C program and the pathname now becomes Proj_name/1/2/2/3. Then this string is converted to the form 1/1/2/2/3, using a simple C program. Finally, this string is then converted by another simple C program to the number 1.1.2.2.3 which is then used to number the box at that level.

Task 3 :

When the number for the box at that particular level has been obtained, it is copied to a temporary file. The contents in the original *text* file are then concatenated to the temporary file using the shell command **cat.** This original *text* file is then renamed as a back up file *text.bak* using the shell command **mv.** The temporary file is then renamed as the *text* file.

The processes of unnumbering the boxes only involved task 1 of the **Numbering Module** and the renaming of the back up file *text.bak* to the original *text* file.

The Verification Module

The processes involved consist of the following two tasks :

Task 1 :

Task 1 deals with the tree traversal in postorder in order to obtain the pathnames of all the nodes under the specific project selected by the user and store them in a temporary file.

Task 2 :

After all the pathnames have been obtained, the labels stored in the *text* files associated with these pathnames are then compared two at a time. This is done using the **cmp** shell command. For instance, if there are five labels which are stored in *text* files of different pathnames, they are compared in the following sequence :
(Let the five labels be stored in *text1*,*text2*,*text3*,*text4*, and *text5*)

Comparison Process :

text1 is compared with	:	text2, text3, text4 and text5
text2 is compared with	:	text3, text4 and text5
text3 is compared with	:	text4 and text5
text4 is compared with	:	text5

The number of comparisons required is 10 (i.e. 4+3+2+1). Generally, if there are n text files, the number of comparisons required will be $(n-1)+(n-2)+(n-3)+\ldots+1$. This shows that the time taken to complete the verification process is dependent on the size of the project.

Data-Flow Module

There are two main processes performed by this module : the display of a DFD on the screen and the printing on paper. The display of a DFD is handled by the two submodules, namely ***Screen Handler*** and ***Text Handler*** of the **Data-Flow Module** discussed in Section 2. The printing process is handled by the submodules ***User Interface***. Basically, the processes performed by these submodules can be classified into the following five tasks:

Task 1 :

This task deals with the travelling of the tree structure in postorder to obtain the particular DFD text file associated with the tree structure so that a complete DFD can be displayed on the screen.

Task 2 :

When the travelling of the tree structure has reached the leaves, the type of DFD text file and level number are obtained so that the type of DFD to be drawn can be determined and then drawn based on the extension

of the filename. The possible DFD symbols allowed to be drawn in a full-screen display are as shown in Figure 15. The possible DFD symbols allowed to be drawn for each level are listed in Table 4.

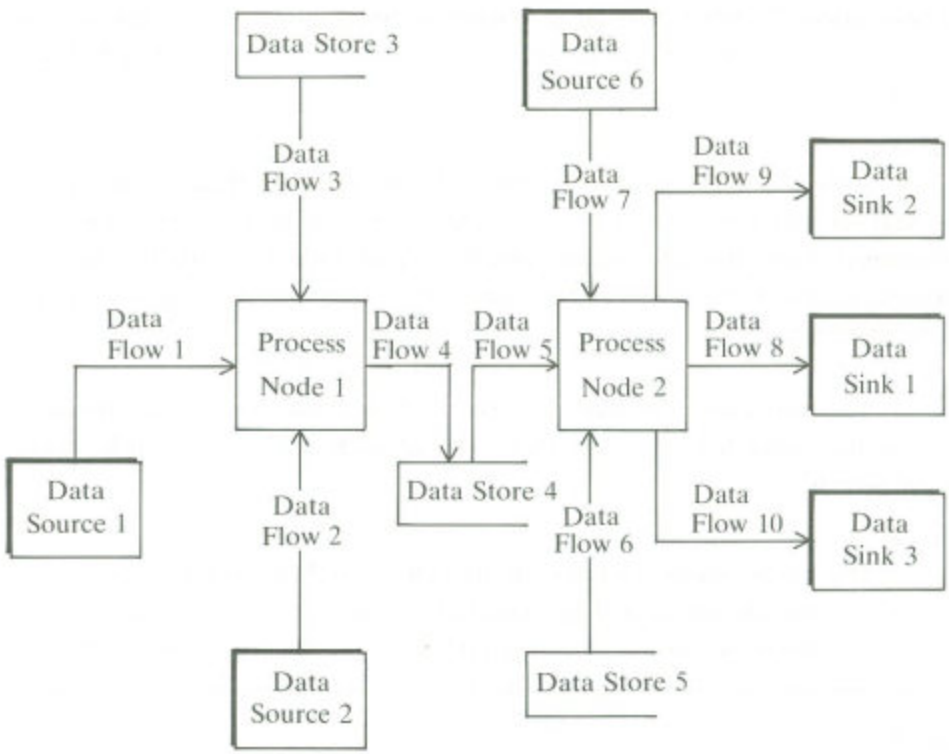


Fig. 15 A Fullscreen Display of Possible DFD Symbols

TABLE 4
POSSIBLE DFD SYMBOLS FOR EACH LEVEL

Level No.	Possible DFD Symbols
1	Data Source, Data Store
2	Data Flow
3	Processing Node
4	Data Flow
5	Data Source, Data Store, Data Sink, Processing Node
6	Data Flow
7	Processing Node, Data Sink
8	Data Flow
9	Data Store, Data Sink

As there is a possibility that there might be more than one DFD symbol being displayed for a particular level, it is crucial that these DFD symbols should not overlap each other. A strategy adopted is to assign a unique drawing position to each of the symbols determined using a set of ***decision tables***[4,16]. When the DFD symbol is being displayed on the screen, it is also at the same time being saved to an array. This task is then followed by task 3.

Task 3 :

After the DFD symbol has been drawn, the ***Text Handler*** will display the text at the proper position associated with the DFD symbol which is determined from the information obtained from task 2. Similarly, the text displayed is also being saved to the same array. Then, the travelling is returned to the parent node.

The processes involved in task 1, task 2 and task 3 are repeated until all the nodes have been travelled. The complete DFD is now fully drawn on the screen.

Task 4 :

The array which contains all the DFD symbols and text are saved to a temporary file which will be renamed by the user when prompted. The file is then moved to the sub-directory/ISDT/Dfd_Dir created to store all DFD files so that they can be manipulated later by the **miscellanea** functions of ISDT.

Task 5 :

Finally, the user can obtain a print out of the DFD by sending the file to the printer. The printing process is performed by the shell command **pr**. If the user has not requested for the file to be saved, the temporary file is then deleted.

Structure-Chart Module

The major processes performed by this module is to enable a user to browse at the whole structure chart with three boxes being displayed at a time and to obtain a print out of the structure chart. The screen display is divided into three portions. The first portion is the display of the boxes with the labels within them and the corresponding joining of vertical lines, horizontal lines and arrows. The second portion is the display of information concerning the boxes displayed in the first portion. The information displayed includes the *level numbers*, the *current node numbers* and the *total number of nodes attached to the nodes currently being displayed*. Besides such information, the boxes displayed in the first portion are represented using '*'s and the nodes attached to these boxes are represented using 'o's. The third portion of the screen is the display of messages. These messages inform the user concerning the valid

arrow keys to press when browsing the other portions of the diagram. These three portions of the screen display are manipulated by the three submodules of the **Structure-Chart Module,** namely ***Screen Handler, Keyboard Handler*** and ***Text Handler,*** discussed in FEATURES OF THE TOOL section. These three portions of the screen display as illustrated in Figure 16 are updated whenever a valid key from the numeric/arrow key pad is pressed. However, if an invalid key is pressed, it is ignored and the screen display remains unchanged.

Note :

The current boxes displayed are node 1 of level 1 and node 1 and 2 of level 2. Level 2 has a maximum of 6 nodes. Node 1 of level 2 consists of 7 nodes while node 2 consists of 2 nodes in their lower levels respectively. The nodes below level 3 are not displayed due to limited screen size.

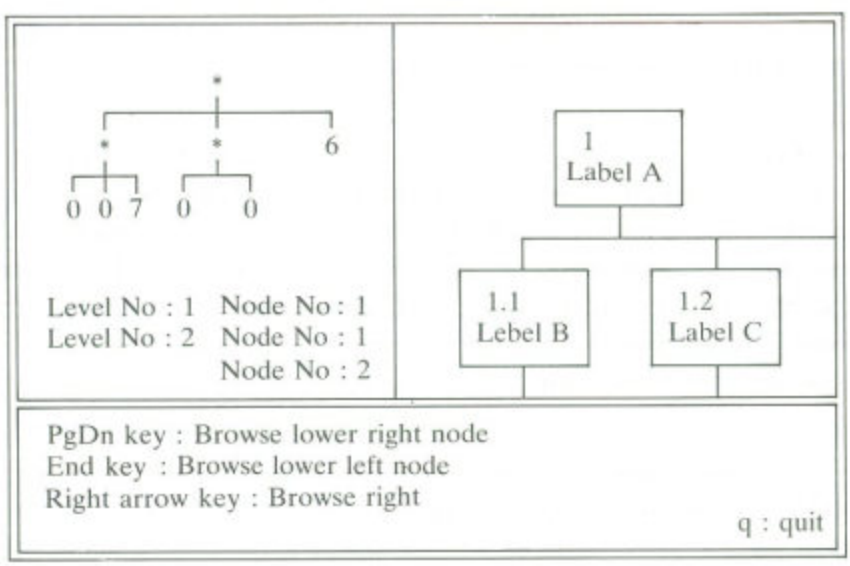


Fig. 16 A Sample Screen Display

The processes involved in the printing of the structure chart can be classified into the following five tasks:

Task 1 :

This task deals with the travelling of the tree structure in postorder. During the travelling process, the pathnames of all the nodes are stored in a temporary file. This temporary file is then manipulated by task 2.

Task 2 :

Task 2 is to remove the string /ISDT/from the pathnames by a simple C program. These pathnames are then split into different levels based on their path lengths, performed by another simple C program. If there are more

than eight pathnames in the file, this file is then split up into two or more files with a maximum of eight pathnames in a file. This breaking down of file contents is also performed by a simple and general C program. The main purpose of this splitting of pathnames is to ensure that the boxes or blocks of these associated pathnames can be displayed within the screen size and thus they will be printed within the paper size.

Task 3 :

Task 3 is the drawing of the boxes or blocks with vertical and horizontal lines, arrows (if requested) and the labels associated with those pathnames displayed on the screen. When these diagrams are being drawn on the screen, they are also being saved in an array of the screen size. The pathname file is then deleted. The contents of the screen array is then written to a temporary file and re-initialized. This drawing process is repeated for the other pathname files. The saving of the screen array of these pathname files are appended to the temporary file one at a time.

Task 4 :

When the drawing process is completed, the user is prompted with a message to confirm the saving of the diagram. If saving is needed, the user is asked to enter a filename. The temporary file is then renamed with this filename and it is then moved to the sub-directory /ISDT/Sc_Dir created to store all the structure charts, hierarchical block diagrams or HIPO diagrams constructed by the user, so that these diagrams can be manipulated later by the miscellanea functions of ISDT.

Task 5 :

This task performs the printing of the structure chart by sending the file which contains the complete diagram to a printer when a print out is requested by the user. If the user has not requested for a saving of the diagram, the temporary file will be deleted.

CONCLUSION

In this paper, a functional description of the different modules of ISDT has been given. This description is then followed by an explanation on the various types of files used. A discussion on the implementation of ISDT is then covered in detail. The different implementation approaches, tools and techniques as well as the developments of selected modules in terms of algo rithms and pseudocodes have been clearly explained. Unlike the CASE tools mentioned in the Introduction section, which are developed on fairly advanced current microcomputer technology, ISDT was developed based on the hierar chical file system of the Xenix OS on the cheapest PC configuration. Conceptually, therefore, the development of ISDT can be seen as an exercise in using modern and established techniques of software development using moderately

priced technology while producing highly powerful features for the ordinary programmer. The differences between ISDT and existing similar tools can be viewed from the different aspects such as the hardware and software requirements, and the major facilities provided by these tools. The existing CASE tools discussed in the INTRODUCTION section are developed based on the workstation technology. These tools can only maximize productivity and efficiency if they are run on workstations equipped with expensive high quality input and output devices. These CASE tools are also developed by a team of experienced software engineers, system analysts and programmers, thus requiring a large amount of investment in manpower and expenditure. As such, the tools are much more directed towards the corporate institutions. The high cost of these tools does not make it easier for the smaller companies who actually need these tools badly.

From the point of view of facilities provided by Excelerator, IEW/WS and IDE, they are very similar to each other. For instance, they provide editors for drawing data flow diagrams, structure charts, and other types of graphs and charts. ISDT in this case is very different from these tools since it only allows users to construct hierarchical block diagrams, data flow diagrams and structure charts. Therefore, the facilities provided by ISDT is unable to compete with these powerful tools although the simple concept being advocated in the development of ISDT can certainly be extended to produce equally useful facilities for ISDT at a later stage. ISDT, however, does have a few similarities with these existing tools. For instance, it provides a facility for numbering the charts which is also a feature in the Excelerator. It also provides a facility for checking the labels of the charts which is similar to these existing CASE tools. Unlike these existing tools, ISDT allows users to enter labels of their own without any restrictions and limitations on the terms used. This is a flexibility which is not provided in these more expensive and sophisticated CASE tools.

An important fact to be mentioned is that these existing tools are rather large software packages, therefore requiring a sizable amount of disk storage. A user who is unfamiliar with these software packages and the more advanced hardware technology will require more time to familiarise with these software packages. These drawbacks are not found in ISDT because ISDT provides facilities that are being used more often rather than having too many complicated things but not used fully. The approach is to offer simple but effective facilities rather than a comprehensive set of facilities which requires detailed training. As a research project, it has thus provided a solution to improve the productivity of a small yet growing team of system developers. It has also given us a beneficial experience in developing a prototype software which will prove to be an open example for the newer generation of software developers.

REFERENCES

1. Ince, D. Software Engineering. The Decade of Change. Peter Peregrinus Ltd., 1986.
2. Shaw, M. Prospects for an Engineering Discipline of Software. *IEEE Software*, Nov., 1990, **7** (6), 15–24.
3. Factor, R.M. and Smith, W.B. A Discipline for Improving Software Productivity. *AT & T Tech. J.*, A Journal of the AT&T Companies., July/August, 1988, 2–9.
4. Pressman, R.S. Software Engineering : A Practitioner's Approach., McGraw–Hill Inc., 1982.
5. Pressman, R.S. Software Engineering : A Practitioner's Approach. International Edition Computer Science Series, 2nd ed., McGraw-Hill Inc., 1987.
6. Datamation. *The Computer-Aided Software Engineering Symposium*, July 1, 1988, p. 58.
7. McClure, C.L. *Review of InTech's Excelerator*, Index Technology Corporation Computer Consultant, Oct. 1984.
8. Information Engineering Workbench/Workstation. KnowledgeWare Inc., 1986.
9. Interactive Development Environments (IDE). Interactive Development Environments Inc., January 1987.
10. Yourdon, E. and Constantine, L.L. *Structured Design : Fundamental of a Discipline of Computer Program and Systems Design*, Prentice Hall Inc., 1979.
11. Peters, L.J. *Software Design : Methods and Techniques*. Yourdon Inc., New York, 1981.
12. Page-Jones, M. *The Practical Guide to Structured System Design*. Yourdon Press, Prentice Hall Inc., 1980.
13. De Marco, T. *Structured Analysis and System Specification*. Yourdon Inc., New York, 1979.
14. XENIX System V Operating System Release Notes, Version 2.1.3. *Introduction to XENIX*. The Santa Cruz Operation Inc., 1986.
15. XENIX System V Operating System Release Notes, Version 2.1.3. *User's Guide*. The Santa Cruz Operation Inc., 1986.
16. Fairley, R.E. *Software Engineering Concepts*. McGraw-Hill Inc., 1985.
17. Wegner, P., Dennis, J., Hammer, M. and Teichroew, D. *Research Directions in Software Technology*. The Massachusetts Institute of Technology, 1979.
18. Jackson, M. *Principles of Program Design*. Academic Press Inc., 1975.
19. Schildt, H. *Advanced C*. McGraw-Hill Inc., 1987.